Programación Avanzada para la Ingeniería
Práctica 1

Programa un conjunto de clases para el manejo de elementos gráficos para JavaFX de forma que se implanten las interfaces:

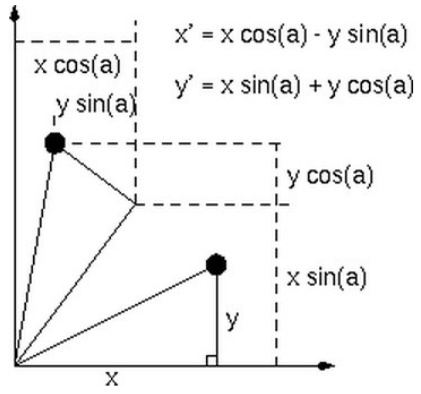
```
interface Dibujable {  // Interfaz para objetos que saben dibujarse

    void dibuja (GraphicsContext gc);
    // Método de dibujado al que se le pasa el objeto gc para la visualización en una ventana

}


interface Transformable {  // Interfaz que implantan clases con transformaciones geométricas

    void traslada(Coordenadas p);  // Traslación en el plano según las coordenadas de p

    void escala(Coordenadas p, double escala);  // Aplicación de un factor de escala con respecto a p

    void rota(Coordenadas p, double angulo); // Rotación con respecto al origen p

}


interface Clonable {  // Interfaz para objetos que saben clonarse

     Object clona();  // Devuelve una copia

}
```

Hay una clase para representar a un punto en 2D:

```
class Coordenadas implements Transformable {  // Representa a un punto 2D

    double x, y;  // Coordenadas del punto

    public static final Coordenadas origenCoordenadas = new Coordenadas();
    // Representa al origen de coordenadas (0,0). Se maneja en un objeto
    // estático y constante

    public Coordenadas() {  // Constructor sin parámetros
        x = y = 0;
    }

    public Coordenadas(double _x) {  // Constructor a partir de coordenada X
        x = _x;
        y = 0;
    }

    public Coordenadas(double _x, double _y) {  // Constructor a partir de dos coordenadas
        x = _x;
        y = _y;
    }

    public Coordenadas(Coordenadas p) {  // Constructor de copia
        x = p.x;
        y = p.y;
    }

    public void traslada(Coordenadas t) {  // Traslación según las coordenadas de t
        x += t.x;
        y += t.y;
    }

    public void escala(Coordenadas posicion, double e) {  // Escalado en posicion con factor e
        x = posicion.x + (x - posicion.x) * e;
        y = posicion.y + (y - posicion.y) * e;
    }
```

```
    public void rota(Coordenadas posicion, double angulo) {  // Rotación en posicion con un ángulo
        double x1, y1, coseno, seno;
        coseno = Math.cos(angulo * Math.PI / 180);
        seno = Math.sin(angulo * Math.PI / 180);
        x1 = (x - posicion.x) * coseno - (y - posicion.y) * seno;
        y1 = (x - posicion.x) * seno + (y - posicion.y) * coseno;
        x = x1 + posicion.x;
        y = y1 + posicion.y;
    }

}
```

La rotación con un ángulo a se puede implantar siguiendo la figura:



Hay una clase base con miembros que van a utilizarse en varias clases derivadas:

```
abstract class ElementoGrafico implements Dibujable, Transformable, Clonable {
// Clase base para elementos gráficos 2D

    protected Color color;  // Color de dibujado
    protected double grosor;  // Grosor de dibujado

    public ElementoGrafico () {  // Constructor sin parámetros
        color = Color.BLACK;  // Dibujado en negro
        grosor = 1;  // Grosor de 1 punto
    }

    public ElementoGrafico (Color c, double g) {  // Constructor a partir de un color y grosor
        color = c;  // Establece color
        grosor = g;  // Establece grosor
    }

    @Override
    public void dibuja (GraphicsContext gc) {
    // Acciones comunes para el dibujado de cualquier elemento gráfico

        gc.setStroke(color);  // Establece color
        gc.setLineWidth(grosor);  // Establece grosor
    }

}
```

A partir de esta clase base se define una clase derivada para representar a una línea:

```
class Linea extends ElementoGrafico {  // Clase Linea derivada de ElementoGrafico

    private Coordenadas origen, fin;  // Línea entre dos puntos

    public Linea () {  // Constructor sin parámetros
        super();  // Inicializa color y grosor
        origen = new Coordenadas();  // Inicializa punto inicial
        fin = new Coordenadas();  // Inicializa el punto final
    }
```

```
    public Linea (Coordenadas p1, Coordenadas p2) {  // Constructor a partir de dos puntos
        super();  // Inicializa color y grosor
        origen = new Coordenadas(p1);  // Inicializa la posición con el primer punto
        fin = new Coordenadas(p2);  // Inicializa el punto final con el segundo
    }

    public Linea (Coordenadas p1, Coordenadas p2, Color c, double g) {
    // Constructor a partir de dos puntos, color y grosor

        super(c, g);  // Establece color y grosor
        origen = new Coordenadas(p1);  // Inicializa la posición con el primer punto
        fin = new Coordenadas(p2);  // Inicializa el punto final con el segundo
    }

    public Linea (Linea l) {  // Constructor de copia
        super(l.color,l.grosor); // Copia color y grosor
        origen = new Coordenadas(l.origen);  // Copia la posición
        fin = new Coordenadas(l.fin);  // Copia el punto final
    }

    @Override
    public ElementoGrafico clona() {   // Crea un clon de este elemento gráfico
        return new Linea(this);  // Utiliza el constructor de copia para crear el clon
    }

    @Override
    public void dibuja (GraphicsContext gc) { // Dibujado de una línea
        super.dibuja(gc);
        gc.strokeLine(origen.x, origen.y, fin.x, fin.y);
    }

    @Override
    public void traslada(Coordenadas t) {  // Traslada una línea según las coordenadas t
        origen.traslada(t);
        fin.traslada(t);
    }

    @Override
    public void escala(Coordenadas posicion, double e) {  // Escala una línea con respecto a una posoción
        origen.escala(posicion, e);
        fin.escala(posicion, e);
    }

    @Override
    public void rota(Coordenadas posicion, double angulo) {  // Rota una línea con respecto a una posición
        origen.rota(posicion, angulo);
        fin.rota(posicion, angulo);
    }

}
```

Define también las clases para rectángulos, circunferencias y arcos:

```
class Rectangulo extends ElementoGrafico {  // Rectángulo en 2D

    Coordenadas[] puntos;  // Coordenadas de los vértices del rectángulo

    public Rectangulo () {  // Constructor sin parámetros
    ...
    }

    public Rectangulo (Coordenadas p, double ancho, double alto) {
        // Constructor para un rectángulo situado en p y con dimensiones ancho y alto
    ...
    }

    public Rectangulo (Coordenadas p, double ancho, double alto, Color c, double g) {
        // Constructor para un rectángulo situado en p y con dimensiones _ancho y _alto, y
        // especificando color y grosor
    ...
    }

    public Rectangulo (Rectangulo r) {  // Constructor de copia entre rectángulos
    ...
    }

    @Override
    public ElementoGrafico clona() {   // Crea un clon de este elemento gráfico
    ...
    }
```

```
    @Override
    public void dibuja(GraphicsContext gc) {  // Dibujado de un rectángulo
    ...
    }

    @Override
    public void traslada(Coordenadas t) {  // Traslación de un rectángulo
    ...
    }

    @Override
    public void escala(Coordenadas p, double e) {  // Escalado de un rectángulo
    ...
    }

    @Override
    public void rota(Coordenadas p, double a) {  // Rotación de un rectángulo
    ...
    }


class Circunferencia extends ElementoGrafico {  // Representa a una circunferencia en 2D

    double radio;  // Radio del círculo
    Coordenadas centro;  // Centro de la circunferencia

    public Circunferencia() {  // Constructor sin datos
    ...
    }

    public Circunferencia(Coordenadas c, double r) {  // Circunferencia en c, con un radio r
    ...
    }

    public Circunferencia(Coordenadas c, double r, Color color, double g) {
    // Circunferencia en c, con un radio r y fijando un color y grosor
    ...
    }

    public Circunferencia(Circunferencia c) {  // Constructor de copia
    ...
    }

    @Override
    public ElementoGrafico clona() {   // Crea un clon de este elemento gráfico
    ...
    }


    @Override
    public void dibuja(GraphicsContext gc) { // Dibujado de una circunferencia
    ...
    }

    @Override
    public void traslada(Coordenadas t) {  // Traslación de una circunferencia
    ...
    }

    @Override
    public void escala(Coordenadas p, double e) {  // Escalado
      ...
   }

    @Override
    public void rota(Coordenadas p, double e) {  // Rotación de una circunferencia
    ...
    }

}


class Arco extends Circunferencia {  // Representa a un arco en 2D

    double anguloInicial, anguloFinal;  // Ángulos en grados

    public Arco () {  // Constructor sin datos
    ...
    }
```

```
    public Arco (Coordenadas p, double _radio, double _anguloInicial, double _anguloFinal) {
        // Arco con centro en un punto, con un radio y ángulos
    ...
    }

    public Arco (Coordenadas p, double _radio, double _anguloInicial, double _anguloFinal, Color c,
        double g) {
        // Arco con centro en un punto, con un radio y ángulos
    ...
    }

    public Arco (Arco a) {  // Constructor de copia
    ...
    }

    @Override
    public ElementoGrafico clona() {   // Crea un clon de este elemento gráfico
    ...
    }

    @Override    // Redefine este método recibido de la clase base
    public void dibuja(GraphicsContext gc) {
    ...
    }

    @Override
    public void rota(Coordenadas p, double a) {
    ...
    }

}
```

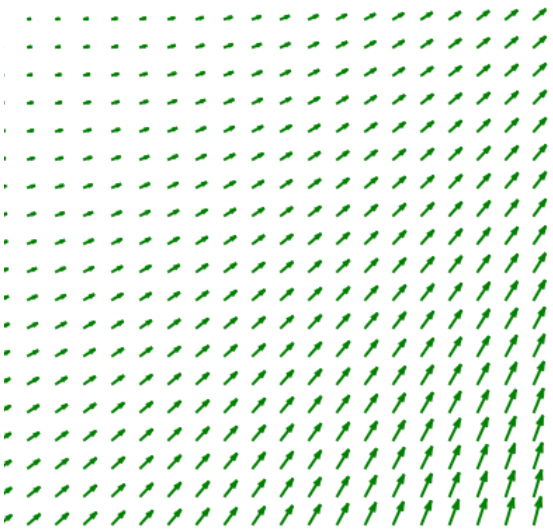
Y una clase para manejar un dibujo como una colección de elementos gráficos:

```
class Dibujo extends ElementoGrafico {  // Un dibujo es también un elemento gráfico

    ArrayList<ElementoGrafico> elementos;  // Un dibujo es una colección de elementos

    public Dibujo() {  // Constructor sin parámetros
    ...
    }

    public Dibujo(Coordenadas _origen) {  // Constructor sin parámetros
    ...
    }

    public Dibujo(Dibujo d) {  // Constructor de copia
    ...
    }

    void carga (ElementoGrafico e) {  // Añade un nuevo elemento gráfico a la colección
    ...
    }

    void cargaCopia (ElementoGrafico e) {  // Añade un clon de un elemento gráfico a la colección
    ...
    }

    @Override
    public void dibuja (GraphicsContext gc) {  // Visualiza todo el dibujo
    ...
    }

    @Override
    public ElementoGrafico clona() {  // Crea un clon de un dibujo
    ...
    }

    @Override
    public void traslada(Coordenadas t) {  // Traslada un dibujo
    ...
    }

    @Override
    public void escala(Coordenadas p, double e) {  // Escala un dibujo
    ...
    }
```

```
    @Override
    public void rota(Coordenadas p, double a) {  // Rota un dibujo
    ...
    }

}
```

De forma que el siguiente método de renderizado genera la siguiente visualización en una ventana:

```
private void renderiza(GraphicsContext gc) {

    Dibujo flecha = new Dibujo();
    flecha.carga(new Linea(new Coordenadas(0, 0), new Coordenadas(10, 0), Color.GREEN, 2));
    flecha.carga((new Linea(new Coordenadas(10, 0), new Coordenadas(7, 1), Color.GREEN, 2)));
    flecha.carga((new Linea(new Coordenadas(10, 0), new Coordenadas(7, -1), Color.GREEN, 2)));
    // Construye un dibujo compuesto por tres líneas para representar a una flecha que apunta hacia la
    // derecha

    Dibujo d = new Dibujo();  // Crea un nuevo dibujo

    for (int x = 0; x < 400; x+=20)  // Repite desde 0 hasta 400 aumentando en 20 unidades
        for (int y = 0; y < 400; y+=20) {  // Repite desde 0 a 400 aumentando en 20 unidades

            Dibujo otraFlecha = new Dibujo(flecha);  // Crea una copia de la flecha

            otraFlecha.escala(Coordenadas.origenCoordenadas, (x + y) / 400.0, (x + y) / 400.0);
            otraFlecha.rota(Coordenadas.origenCoordenadas, (x + y) / 10.0);
            otraFlecha.traslada(new Coordenadas(x, y));
            // La escala, la rota y la traslada

            d.carga(otraFlecha);  // La carga en el dibujo
        }

    d.dibuja(gc);

}
```



Seguidamente se muestra otro ejemplo:

```
    private void renderiza(GraphicsContext gc) {

        Coordenadas origen = new Coordenadas(200, 200);
        // Punto de referencia para realizar transformaciones

        Dibujo d = new Dibujo();
        d.carga(new Linea(new Coordenadas(0, 0), new Coordenadas(0, 100), Color.GREEN, 2));
        d.carga(new Circunferencia(new Coordenadas(0, 120), 20, Color.BLUE, 3));
        // Crea un dibujo en el que carga una línea vertical y una circunferencia en su extremo superior

        d.traslada(origen);
        // Lo traslada al punto de referencia
```

```
        Dibujo grande = new Dibujo();
        for(int i = 0; i < 8; i++) {
            d.rota(origen, 45);
            grande.cargaCopia(d);
        }
        // Crea un nuevo dibujo donde carga 8 copias del primero rotado con respecto a la referencia

        grande.dibuja(gc);  // Visualiza el dibujo

        Dibujo pequeno = new Dibujo(grande);  // Crea una copia de este dibujo

        pequeno.escala(origen, 0.5);
        pequeno.rota(origen, 22.5);
        pequeno.dibuja(gc);
        // Lo rota, escala y dibuja

    }
```